AD-A197 674

AUTOCREW IMPLEMENTATION:

INBOUND SURFACE-TO-AIR MISSILE SIMULATION

Brenda L. Belkin

Laboratory for Control & Automation
Department of Mechanical & Aerospace Engineering
Princeton University

May 1988

CONTENTS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) AUTOCREW IMPLEMENTATION: INBOUND SURFACE-TO-AIR MISSILE SIMULATION		5. TYPE OF REPORT & PERIOD COVERED INTERIM TECHNICAL REPORT
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Brenda L. Belkin		8. CONTRACT OR GRANT NUMBER(s) DAAG29-84-K-0048 20155-MA
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Mechanical & Aerospace Engineering PRINCETON UNIVERSITY, PRINCETON, NJ 08544		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE May 1988
		13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence; Control Theory; Flight Control Systems, Computer Software; Cybernetics.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE Attached

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Nine multiple cooperating rule-based systems for the combat aircraft environment were developed and implemented in the AUTOCREW expert system. Each component rule-based system is modelled on a typical World War II bomber crew member having specific task responsibilities. The bases for modelling integrated rule-based systems as crew members are twofold. First, tasks performed by crew members are easily identified, well-defined, and familiar to operations personnel and aircraft system designers. The issue of task familiarity is essential from a human engineering standpoint. For example, the pilot of a single-seat combat aircraft categorizes the tasks he performs into logical groups such as navigation, flight control, systems monitoring, failure diagnosis, and fire control operations. Implementing an AI-based pilot aid as a logical extension of human operations is highly desirable. The pilot must identify with the task performed by the on-board aid, so a crew-model design of component knowledge bases is commensurate with the pilot's experience and training.

AUTOCREW IMPLEMENTATION:
INBOUND SURFACE-TO-AIR MISSILE SIMULATION

1. Introduction

Nine multiple cooperating rule-based systems for the combat aircraft environment were developed and implemented in the AUTOCREW expert system [1]. Each component rule-based system is modelled on a typical World War II bomber crew member having specific task responsibilities. The bases for modelling integrated rule-based systems as crew members are twofold. First, tasks performed by crew members are easily identified, well-defined, and familiar to operations personnel and aircraft system designers. The issue of task familiarity is essential from a human engineering standpoint. For example, the pilot of a single-seat combat aircraft categorizes the tasks he performs into logical groups such as navigation, flight control, systems monitoring, failure diagnosis, and fire control operations. Implementing an AI-based pilot aid as a logical extension of human operations is highly desirable. The pilot must idetify with the task performed by the on-board aid, so a crew-model design of component knowledge bases is commensurate with the pilot's experience and training.

The AUTOCREW crew members are responsible for performing tasks and controlling functions associated with the aircraft and on-board systems. The modelled crew members are COPILOT (flight control), ENGINEER (system diagnosis, reconfiguration), NAVIGATOR (navigation), COMMUNICATOR (radio/data operations), OBSERVER (lookout and alarm) ATTACKER (offensive weapon control), DEFENDER (defensive weapon control), and SPOOFER (countermeasures). The ninth rule-based system (EXECUTIVE) functions on a higher level than the preceding, more task-specific systems. The EXECUTIVE coordinates mission-specific tasks and has knowledge of the mission plan. The human Pilot acts in the capacity of mission coordinator. He monitors the AUTOCREW systems and attends to mission planning and problems brought to his attention by the AUTOCREW components.

2. Development of Rule-Based System Cooperation

The AUTOCREW knowledge bases are developed and implemented independently. Shared parameters are identified throught the design process. These "global" variables are used to exchange information between knowledge bases. A change in value of a shared parameter in one knowledge base can invoke search activity within another knowledge base. Hence, dynamic changes in the aircraft environment are reflected and handled appropriately within each AUTOCREW knowledge base.

3. Knowledge-Base Development and Implementation

3.1 Knowledge-Base Development

Before implementing a knowledge base in the development environment, a graphical representation of the structure is formed, thereby providing a first glance overview of the knowledge base. This greatly facilitates modification of the knowledge base structure and content. A graphical representation of the DEFENDER's knowledge base is shown in Fig. 1. Rectangular boxes represent knowledge-base parameters and ovals represent rules. Parameters describe factual information about the domain, whereas rules describe the procedural relationships between parameters. The "state" of a parameter is denoted by its value. Parameter values reside in slots within a parameter box. Lines connecting parameters via rules form the procedural relationships between parameters. An AND relationship, denoted by a joining arc between parameters is shown n Rule D005. An example of an OR relationship is shown in Rule D001; this rule is read

IF	the value of the parameter INBOUND ENEMY is NONE
OR	the value of the parameter DEFENDER SUCCESS has been determined
THEN	set the value of the parameter DEFENDER SEARCH COMPLETED to TRUE

The basis for cooperative activity among rule-based systems is the concept of shared parameters. The symbol "S" denotes shared parameters in the DEFENDER's knowledge base shown in Fig.1. The parameters INBOUND ENEMY and INBOUND DIRECTION are shared parameters, residing in the DEFENDER and OBSERVER knowledge bases. The OBSERVER supplies these parameters through assessment of the external environment. The DEFENDER uses the parameters to decide when activation of the defensive fire control system is warranted. The remaining seven knowledge bases also use these parameters to decide when additional tasks need to be performed.

In order to simulate a realistic, dynamic environment in which the rule-based systems operate, the values of several parameters may be changed at any time during the simulation. These parameters are denoted by the symbol "D" in Fig. 1. The designer testing the knowledge base changes the parameter values to test the search logic flow. In an

operational system, the values of these parameters would be determined from sensor measurements.

The provision for user interaction with a dynamic rule-based system is necessary in order to allocate full control of the software framework to the user. Parameters that must have their values supplied by the user are denoted in Fig.1 by shaded areas surrounding the lower portions of parameter boxes. For example, Rule D007 is interpreted as

> "determine the value of the parameter PILOT SELECTS WEAPON by asking the user to provide its value. When a value has been assigned, set the parameter DEFENDER WEAPON SELECTED to TRUE."

Rule D007 also demonstrates how the search process is controlled utilizing "don't care" or "fuzzy" parameter states. The premise of Rule D007 is satisfied when any value of the parameter PILOT SELECTS WEAPON is determined.

3.2 Knowledge-Base Implementation

The Princeton Rule-Based Controller (PRBC) is a unique software architecture for combining procedural and symbolic processing for rule-based system development [2]. The knowledge base is developed in the IQLISP computer language and is translated with the inference engine into the Turbo PASCAL computer language. The PRBC is a highly flexible development tool; numerical code is developed in the naturally procedural language PASCAL, and the knowledge base is developed in the naturally symbolic language LISP. The ease with which PASCAL code can be embedded in the knowledge base structure makes the PRBC a highly suitable tool for task and algorithm scheduling.

Listing 1 shows how the graphical representation of the DEFENDER knowledge base in Fig. 1 is implemented using PRBC syntax. Each rule contains PREMISE (if) and ACTION (then) statements that describe the parameter rule relations depicted in the graphical representation. What is not shown in the knowledge base of Fig. 1 is embedded PASCAL code contained in the PREMISE and ACTION statements of the rules. For example, Rule D006 in Listing 1 illustrates how calls to the PASCAL routines "trckng_scnng_enmy;", "fire_cntrl_lckd_on;" and "enmy_trjctry_computed;" are defined within the rule's action. The PASCAL procedures are ignored during knowledge base testing in LISP. When the knowledge base is translated, calls to the PASCAL procedures are scheduled using search. The search logic can be tested in both the LISP and PASCAL development environments. The PRBC Development System inference

engine uses goal-directed, depth-first search to operate on a knowledge base.

4. Simulation Testbed And Results

4.1 AUTOCREW Simulator

The simulation testbed for AUTOCREW resides on an IBM PC-AT computer. In order to simulate the dynamic flight environment realistically, an interactive testbed was devised using the computer keyboard as the simulation controller. Coded keystrokes control the values of several knowledge-base parameters, thereby allowing the designer to test search activity interactively by changing parameter values. Hence, the designer can control and verify the knowledge-base logic flow and change the simulation on-line. This method is most representative of the actual dynamic flight environment. When all simulation possibilities have been tried, the designer can make suitable adjustments to the knowledge base and continue.

Interaction with AUTOCREW is via the keyboard and graphics display. The AUTOCREW display as shown during an inbound SAM simulation is depicted in Figure 2. Shown are nine window areas for each AUTOCREW member. Messages are sent to the Pilot from AUTOCREW and displayed in the appropriate window area. The most useful messages during preliminary design of multiple cooperating expert systems are those that tell the designer which tasks would be performed in the fully developed system. When an AUTOCREW member requires attention, the window area assigned to the AUTOCREW member flashes. Another feature of the Pilot/AUTOCREW graphics interface are controlled scrolling of old information. Although an operational AUTOCREW would surely use graphical representation of AUTOCREW activities and data, the developed testbed gives the designer a good idea of crew member cooperation and Pilot/AUTOCREW interaction in the early development stages.

4.2 Inbound Surface-to-Air Missile Example

A description of the AUTOCREW simulation shown in Fig. 2 demonstrates system cooperation. Initially, the OBSERVER reports "all clear" as external sensors have not detected anything. The message is repeated upon each sensor sweep until a simulated SAM detection occurs. Then, the OBSERVER's parameter INBOUND ENEMY changes from NONE to SAMS. In the simulation, the inbound direction displayed with the "SAMS!!" message is a randomized integer value of the OBSERVER's parameter INBOUND DIRECTION. The SAMS value of INBOUND ENEMY triggers search activity within the DEFENDER knowledge base. The DEFENDER asks the Pilot to arm

the defensive fire control sequence. The DEFENDER then engages the tracking radar; the fire control system locks onto the SAM and computes the expected SAM trajectory. The COMMUNICATOR reports the SAM attack to the Pilot's wingmates and the control base. The DEFENDER calculates the circular error probables (CEP) of the on-board defensive weapons to find the best one for SAM destruction. In addition, the DEFENDER attempts to classify the SAM using signal information obtained from the OBSERVER and its own sensors. Knowledge of the SAM classification and defensive weapon CEPs aids in the DEFENDER's selection of the best and next-best weapons. Once found, the Pilot is asked to confirm the recommended weapon (MISSILE #1). If the Pilot inputs NO, then the next-best weapon is presented. In the event that the Pilot has armed the system and not disagreed with the selection within a given time, the DEFENDER selects the weapon and proceeds with the fire control sequence.

While the DEFENDER performs these tasks, the COPILOT defines and performs an evasive maneuver, subject to the Pilot's approval. The SPOOFER selects and deploys countermeasures appropriate to a SAM, and the OBSERVER attempts to determine the SAM launch site. The COMMUNICATOR displays SAM confirmation messages from the Pilot's wingmates. The ENGINEER continues to monitor the aircraft systems and evaluates the aircraft's capabilities for the COPILOT. The NAVIGATOR locates the nearest friendly air bases and searches for friendly aircraft to provide possible fire support against the SAM site. Since the aircraft is proceeding to the target area as planned, the ATTACKER continues preparing for mission target engagement. The EXECUTIVE's responsibilities are to analyze the current mission status and prioritize and coordinate AUTOCREW tasks. The EXECUTIVE orders additional AUTOCREW tasks to assist in mission status assessment and in achieving mission goals.

The messages in the DEFENDER's display area "trckng_scnng_enmy;", "fire_cntrl_lckd_on;", "enmy_trjctry_cmpt;" and "computing_cep;" are PASCAL procedures that would control and perform their implied tasks. The PASCAL routines that perform these tasks remain to be developed. Hence, the cooperating rule-based system designer gets a realistic overview of AUTOCREW functions before developing the details.

5. CONCLUSIONS

Nine cooperating rule-based systems for aircraft control were implemented using the Princeton Rule-Based Controller (PRBC) Development System on single-processor IBM PC-AT computer. Each component knowledge base of

AUTOCREW was modelled as a typical World War II bomber crew member performing routine and specialized tasks. The AUTOCREW organizational approach facilitates the breakdown and distribution of flight tasks in a consistent and natural manner, amenable to the designer's experience. Logical task distribution aids in knowledge-base development and in identifying cooperative tasks in a multiple rule-based system.

Many advantages can be drawn from global implementation within a multiple-system framework. Using the PRBC development tool and an interactive simulation testbed, it was demonstrated that an effective simulation of pilot-system interaction and intersystem cooperation could be obtained. It also was shown that minimal knowledge-base details were needed to implement realistic simulations. These simulations enable the designer to identify areas of extensive system cooperation and to plan future individual knowledge-base requirements more efficiently.

6. REFERENCES

[1] B. L. Belkin and R.F. Stengel, "Cooperative Rule-Based Systems for Aircraft Control, in *Proceedings of the 26th IEEE Conference on Decision and Control*, 1987, pp.1934-1940.

[2] D.A. Handelman and R.F. Stengel, "A Theory for Fault-Tolerant Flight Control Combining Expert System and Analytical Redundancy Concepts", in *Proceedings of the 1986 AConference*, 1986, pp. 375-384.

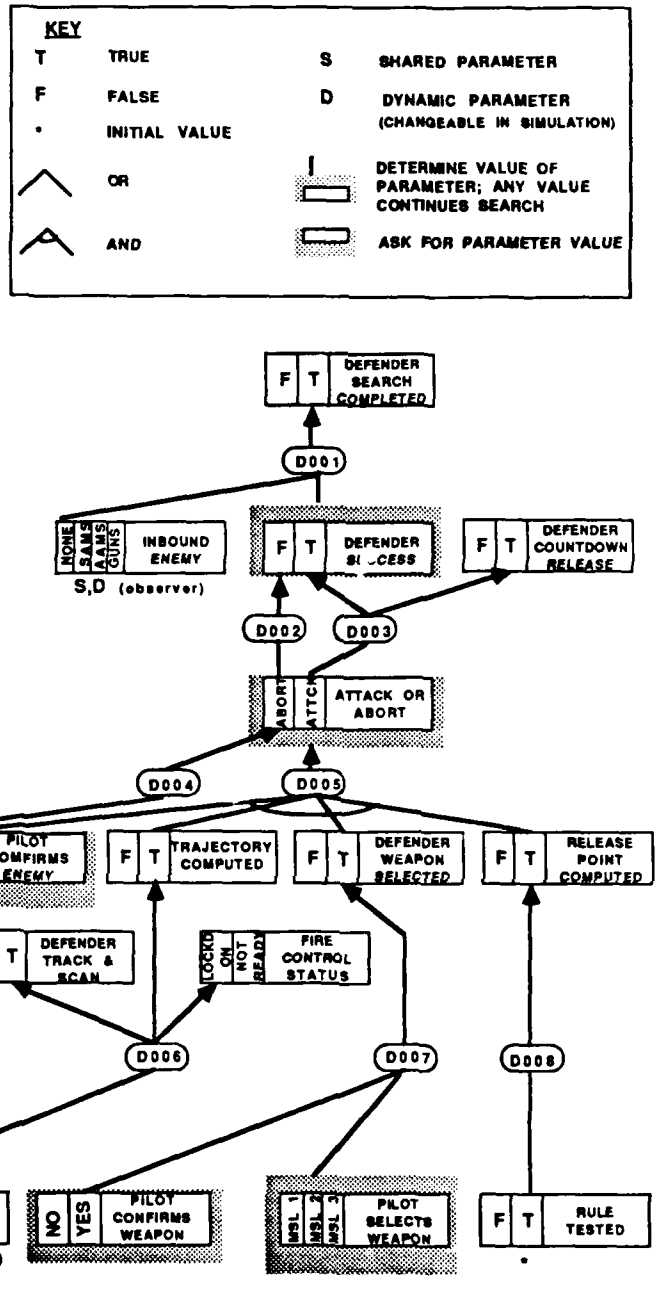


Figure 1. Skeletal Knowledge Base of AUTOCREW Member DEFENDER

Listing 1. Skeletal DEFENDER Knowledge Base

```
[RULE_D001
 [PREMISE '
  ($OR
    ($EQ INBOUND_ENEMY 'NONE)
    ($DETERMINE_VALUE_OF DEFENDER_SUCCESS))]

 [ACTION '($SETQ DEFENDER_SEARCH_COMPLETED 'TRUE)]]


[RULE_D002
 [PREMISE '($EQ ATTACK_OR_ABORT 'ABORT)]

 [ACTION '($SETQ DEFENDER_SUCCESS FALSE)]]

[RULE_D003
 [PREMISE '($EQ ATTACK_OR_ABORT 'ATTACK)]

 [ACTION '(($PASCAL "cntng_dwn_rlsng_wpn;")
       ($SETQ DEFENDER_COUNTDOWN_RELEASE 'TRUE)
       ($SETQ DEFENDER_SUCCESS 'TRUE))]]

[RULE_D004
 [PREMISE '($EQ PILOT_CONFIRMED_ENEMY 'NO)]

 [ACTION '($SETQ ATTACK_OR_ABORT 'ABORT)]]


[RULE_D005
 [PREMISE '
  ($AND
    ($EQ PILOT_CONFIRMED_ENEMY 'YES)
    ($EQ TRAJECTORY_COMPUTED 'TRUE)
    ($EQ DEFENDER_WEAPON_SELECTED 'TRUE)
    ($EQ RELEASE_POINT_COMPUTED 'TRUE))]

 [ACTION '(($PASCAL "ask_for_parameter('attack/abort',
              attack_or_abort,defender);")
       ($SETQ ATTACK_OR_ABORT ATTACK_OR_ABORT))]]

[RULE_D006
 [PREMISE '($NOT ($EQ INBOUND_DIRECTION '0))]

 [ACTION '(($PASCAL "trckng_scnng_enmy;
              fire_cntrl_lckd_on;
              enmy_trjctry_computed;")
       ($SETQ DEFENDER_TRACK_SCAN 'TRUE)
       ($SETQ FIRE_CONTROL_STATUS 'LOCKED_ON)
       ($SETQ TRAJECTORY_COMPUTED 'TRUE))]]

[RULE_D007
 [PREMISE '(($PASCAL "computing_cep;")
    ($OR
       ($EQ PILOT_CONFIRMS_WEAPON 'YES)
       ($DETERMINE_VALUE_OF PILOT_SELECTS_WEAPON)))]

 [ACTION '($SETQ DEFENDER_WEAPON_SELECTED 'TRUE)]]

[RULE_D008
 [PREMISE '($EQ RULE_TESTED 'TRUE)]

 [ACTION '(($PASCAL "cmptng_rlease_pnt;")
       ($SETQ RELEASE_POINT_COMPUTED 'TRUE))]]
```

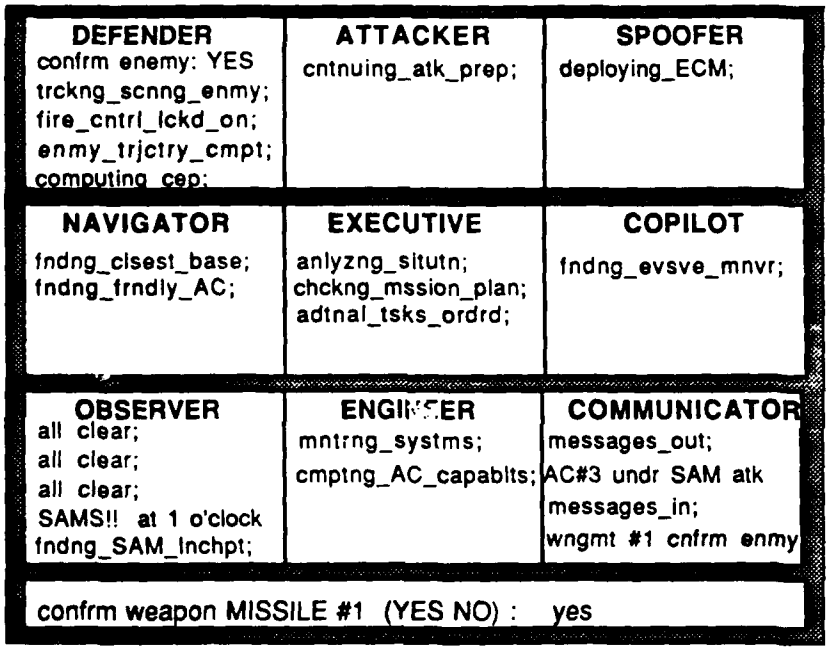


Figure 2. Display of Inbound SAM Simulation At Detection Time

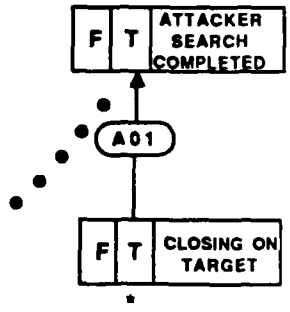
7. AUTOCREW Knowledge Base Listings For Inbound SAM Attack Simulation

The nine AUTOCREW member knowledge bases are found in the following sections:

The DEFENDER's knowledge base has been slightly modified from the one discussed in section 5. Some rules have been added and some have been reordered. These changes do not in any way change the simulation described in section 5.

Each knowledge base has been trimmed from a larger parent knowledge base so that only the decision branch relevant to the inbound SAM simulation remains. This facilitates a clear overview of the knowledge-base functions and implementation methodology. A graphical representation of each knowledge base precedes each implemented code structure.

Figure 7.1 Attacker Knowledge Base



Listing 7.1 Attacker Knowledge Base

```
~  attacker kb for SAM attack demo
~  assumes fighter is closing in on target when SAM is released
~
~
~
~ (SETQ *SYSTEM_ID_CHARACTER* "A4")
~ (SETQ *EXPERT_ID* "attacker")

~  attacker  parameters


 [INTERN_SYMBOL_GROUP *ATTACKER_PARAMETERS*

     [ATTACKER_SEARCH_COMPLETED
                  (EXPECT 'BOOLEAN)]

     [CLOSING_ON_TARGET
                  (EXPECT 'BOOLEAN)
                  (INITIAL_VALUE 'TRUE)]

  ]


~  attacker rules

  [INTERN_SYMBOL_GROUP *ATTACKER_RULES*

     [RULE_A01
       [PREMISE '($EQ CLOSING_ON_TARGET 'TRUE)]
       [ACTION  '( ($PASCAL "cntnuing_atk_prep;")
                   ($SETQ ATTACKER_SEARCH_COMPLETED 'TRUE))]
     ] ]

~ (FIND_PARAMETER_RULE_RELATIONS *ATTACKER_PARAMETERS*  *ATTACKER_RULES*)
```

Listing 7.2 Defender Knowledge Base

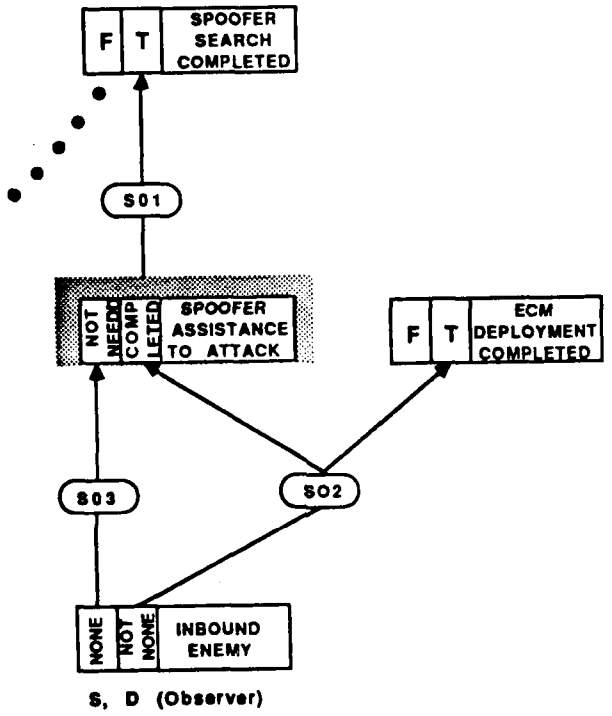
```
~  defender kb for SAM attack simulation
~  assumes that SAM defense is successful (i.e SAM destroyed)
~
~  shared parameters (with executive) are LAUNCH_CLEARANCE
~                                        SUCCESS_DETERMINED
~                    (with observer)     INBOUND_ENEMY
~                                        INBOUND_DIRECTION
~
~    also top level parameters of all eight remaining knowledge bases
~

~ (SETQ *SYSTEM_ID_CHARACTER* "D4")
  (SETQ *EXPERT_ID* "defender")

~    defender parameters

 [INTERN_SYMBOL_GROUP *DEFENDER_PARAMETERS*

   [DEFENDER_SEARCH_COMPLETED
                                (EXPECT 'BOOLEAN)]

   [DEFEND_SUCCESS
                                (EXPECT 'BOOLEAN)]

   [DEFEND_COUNTDOWN_RELEASE
                                (EXPECT 'BOOLEAN)]

   [ATTACK_OR_ABORT
                                (EXPECT '(ATTACK  ABORT))]

   [TRAJECTORY_COMPUTED
                                (EXPECT 'BOOLEAN)]

   [DEFEND_WEAPON_SELECTED
                                (EXPECT 'BOOLEAN)]

   [WEAPON_RECOMMENDED
                                (EXPECT '(NONE MSSL_1 MSSL_2 MSSL_3 MM20_GUNS))]

   [DEFEND_RELEASE_POINT_COMPUTED
                                (EXPECT 'BOOLEAN)]

   [PILOT_CONFIRMED_ENEMY
                                (EXPECT '(YES NO))]

   [DEFEND_FIRE_CONTROL_STATUS
                                (EXPECT '(LOCKED_ON NOT_READY))]

   [DEFEND_TRACK_SCAN
                                (EXPECT 'BOOLEAN)]

   [PILOT_CONFIRMS_WEAPON
```

```
                                    (EXPECT '(YES NO) )]

  [PILOT_SELECTS_WEAPON
                                    (EXPECT '(NONE MSSL_1 MSSL_2 MSSL_3 MM20_GUNS))]


  [RULE_TESTED
                                    (EXPECT 'BOOLEAN)
                                    (INITIAL_VALUE 'TRUE)]

]

~ defender rules

  [INTERN_SYMBOL_GROUP *DEFENDER_RULES*


  [RULE_D01
    [PREMISE  '($OR ($EQ INBOUND_ENEMY 'NONE)
                    ($DETERMINE_VALUE_OF DEFEND_SUCCESS))]

    [ACTION  '(
      ($SETQ DEFENDER_SEARCH_COMPLETED 'TRUE)
      ($PASCAL
        "initialize_buffers(defender);
         off_screen(defender);
         start_up_display;")
              )] ]


  [RULE_D02
    [PREMISE '($EQ ATTACK_OR_ABORT 'ATTACK)]

     [ACTION  '(
       ($SETQ DEFEND_COUNTDOWN_RELEASE 'TRUE)
       ($PASCAL
         "cntng_dwn_rlsng_wpn;")
       ($SETQ DEFEND_SUCCESS 'TRUE)
       ($UNSETQ SUCCESS_DETERMINED)
       ($DETERMINE_VALUE_OF SUCCESS_DETERMINED)
       ($SETQ INBOUND_ENEMY 'NONE)
                )] ]



  [RULE_D03
    [PREMISE '($OR ($EQ INBOUND_ENEMY 'NONE)
                   ($EQ ATTACK_OR_ABORT 'ABORT)) ]

    [ACTION  '( ($SETQ DEFEND_COUNTDOWN_RELEASE 'FALSE)
                ($SETQ DEFEND_SUCCESS 'FALSE))] ]


  [RULE_D04
    [PREMISE '($AND ($EQ PILOT_CONFIRMED_ENEMY 'YES)
                    ($EQ TRAJECTORY_COMPUTED 'TRUE)
```

```
                    ($EQ DEFEND_WEAPON_SELECTED 'TRUE)
                    ($EQ DEFEND_RELEASE_POINT_COMPUTED 'TRUE) )]
   [ACTION '(
     ($PASCAL
       "ask_for_parameter(''attack/abort '',attack_or_abort,defender);")
     ($SETQ ATTACK_OR_ABORT
        ($PASCAL  " p[ATTACK_OR_ABORT].s_value"))) ] ]


[RULE_D05
  [PREMISE '($EQ PILOT_CONFIRMED_ENEMY 'NO)]
  [ACTION  '($SETQ ATTACK_OR_ABORT 'ABORT)] ]


[RULE_D06
  [PREMISE '($EQ RULE_TESTED 'TRUE)]

  [ACTION  '(
    ($DETERMINE_VALUE_OF ENGINEER_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF SPOOFER_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF NAVIGATOR_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF COMMUNICATOR_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF OBSERVER_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF COPILOT_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF ATTACKER_SEARCH_COMPLETED)
    ($DETERMINE_VALUE_OF EXECUTIVE_SEARCH_COMPLETED)
    ($PASCAL
       "ask_for_parameter(''cnfrm enemy '',pilot_confirmed_enemy,
                  defender);")
     ($SETQ PILOT_CONFIRMED_ENEMY
        ($PASCAL
          "p[PILOT_CONFIRMED_ENEMY].s_value"))
            )] ]


[RULE_D07
  [PREMISE '($NOT ($EQ INBOUND_DIRECTION '0)) ]

  [ACTION  '(
    ($PASCAL
      "trckng_scnng_enmy;")
    ($SETQ DEFEND_TRACK_SCAN 'TRUE)
    ($PASCAL
      "fire_cntrl_lckd_on;")
    ($SETQ DEFEND_FIRE_CONTROL_STATUS 'LOCKED_ON)
    ($PASCAL
      "enmy_trjctry_computed;
       calculating_cep;")
    ($SETQ TRAJECTORY_COMPUTED 'TRUE)
    ($DETERMINE_VALUE_OF LAUNCH_CLEARANCE) )] ]


 [RULE_D08
   [PREMISE '($EQ RULE_TESTED 'TRUE)]
   [ACTION '(
```

```
      ($SETQ WEAPON_RECOMMENDED
        ($PASCAL "p[WEAPON_RECOMMENDED].s_value"))
      ($PASCAL
         "ask_for_parameter(''cnfrm weapon '' +
            symbol_name[p[WEAPON_RECOMMENDED].s_value],
            pilot_confirms_weapon,defender);")
       ($SETQ PILOT_CONFIRMS_WEAPON
          ($PASCAL
            "p[PILOT_CONFIRMS_WEAPON].s_value"))
            )] ]


[RULE_D09
   [PREMISE '($OR ($EQ PILOT_CONFIRMS_WEAPON 'YES)
                  ($DETERMINE_VALUE_OF PILOT_SELECTS_WEAPON)) ]

   [ACTION  '($SETQ DEFEND_WEAPON_SELECTED 'TRUE)] ]



 [RULE_D10
  [PREMISE '($EQ RULE_TESTED 'TRUE)]

  [ACTION  '(
     ($PASCAL
       "ask_for_parameter(''your choice'',pilot_selects_weapon,defender);")
      ($SETQ PILOT_SELECTS_WEAPON
         ($PASCAL
           "p[PILOT_SELECTS_WEAPON].s_value"))
            )] ]


[RULE_D11
  [PREMISE '($EQ RULE_TESTED 'TRUE)]

  [ACTION  '(
     ($PASCAL
       "cmptng_release_pnt;")
     ($SETQ DEFEND_RELEASE_POINT_COMPUTED 'TRUE))] ] ]


(FIND_PARAMETER_RULE_RELATIONS *DEFENDER_PARAMETERS* *DEFENDER_RULES*)
```

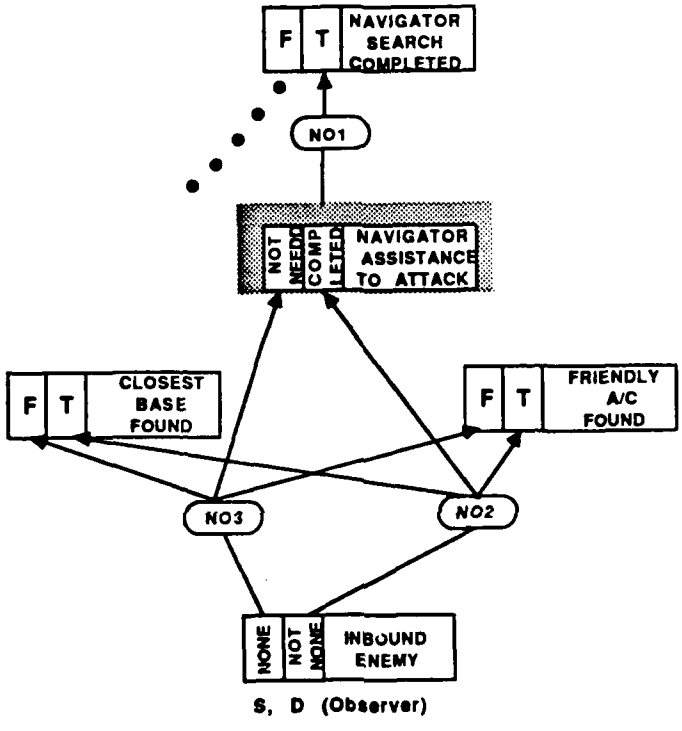
Figure 7.3 Spoofer Knowledge Base



Listing 7.3 Spoofer Knowledge Base

```
~  spoofer kb for SAM attack demo
~
~  shared parameters (with defender/observer) INBOUND_ENMEY
~


~ (SETQ *SYSTEM_ID_CHARACTER* "S4")
~ (SETQ *EXPERT_ID* "spoofer")

~   spoofer  parameters


 [INTERN_SYMBOL_GROUP *SPOOFER_PARAMETERS*

     [SPOOFER_SEARCH_COMPLETED
                  (EXPECT 'BOOLEAN)]

     [SPOOFER_ASSISTANCE_TO_ATTACK
                  (EXPECT '(NOT_NEEDED, COMPLETED)]

     [ECM_DEPLOYMENT_COMPLETED
                  (EXPECT 'BOOLEAN)]

  ]

~   spoofer rules

  [INTERN_SYMBOL_GROUP *SPOOFER_RULES*

     [RULE_S01

        [PREMISE '($DETERMINE_VALUE_OF SPOOFER_ASSISTANCE_TO_ATTACK)]
        [ACTION  '($SETQ SPOOFER_SEARCH_COMPLETED 'TRUE)]
     ]


     [RULE_S02

         [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
         [ACTION  '(
             ($PASCAL
               "fndng_apprp_ECM;
                deploying_ECM;")
             ($SETQ ECM_DEPLOYMENT_COMPLETED 'TRUE)
             ($SETQ SPOOFER_ASSISTANCE_TO_ATTACK 'COMPLETED))] ]


     [RULE_S03

         [PREMISE '($EQ INBOUND_ENEMY 'NONE)]
         [ACTION  '($SETQ SPOOFER_ASSISTANCE_TO_ATTACK 'NOT_NEEDED)] ]

  ]


~   (FIND_PARAMETER_RULE_RELATIONS *SPOOFER_PARAMETERS*  *SPOOFER_RULES*)
```

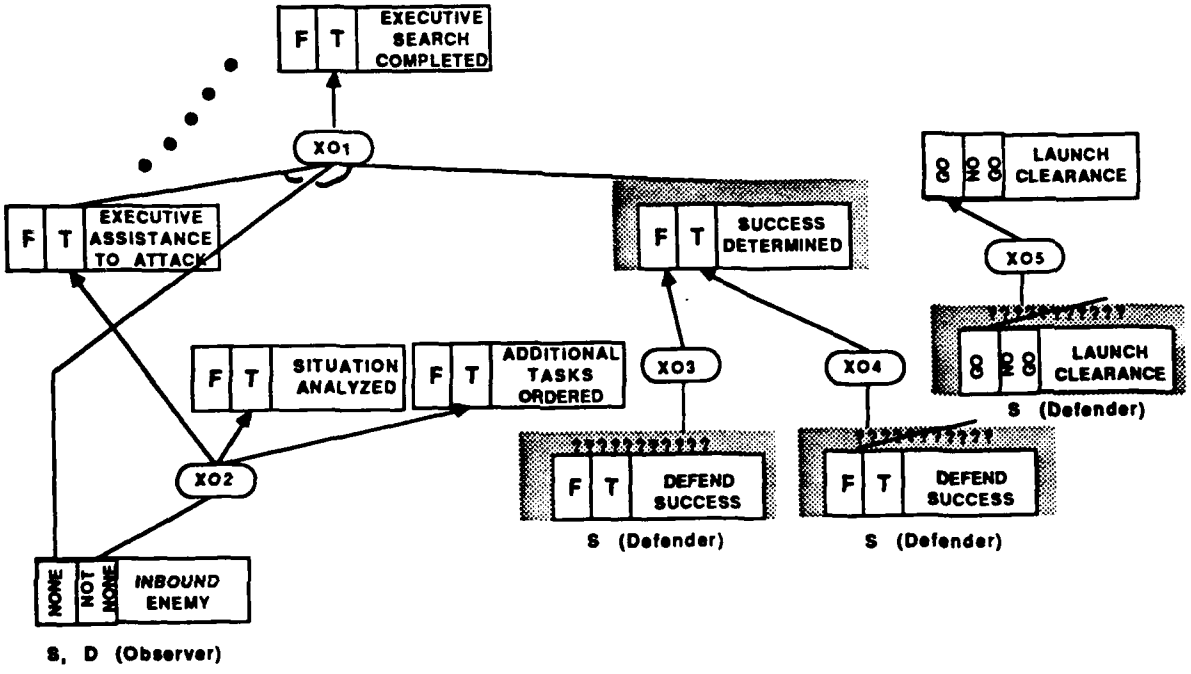
Figure 7.4 Navigator Knowledge Base



Listing 7.4 Navigator Knowledge Base

```
~  navigator kb for SAM attack demo
~
~  shared parameter (with defender/observer) is INBOUND_ENEMY
~

~  (SETQ *SYSTEM_ID_CHARACTER* "N4")
~ (SETQ *EXPERT_ID* "navigator")

~  navigator  parameters

 [INTERN_SYMBOL_GROUP *NAVIGATOR_PARAMETERS*

    [NAVIGATOR_SEARCH_COMPLETED
                (EXPECT 'BOOLEAN)]

    [NAVIGATOR_ASSISTANCE_TO_ATTACK
                (EXPECT '(NOT_NEEDED, COMPLETED)]

    [CLOSEST_BASE_FOUND
                (EXPECT 'BOOLEAN)]

    [FRIENDLY_AC_FOUND
                (EXPECT 'BOOLEAN)]

  ]

~  navigator rules

  [INTERN_SYMBOL_GROUP *NAVIGATOR_RULES*

    [RULE_N01

       [PREMISE '($DETERMINE_VALUE_OF NAVIGATOR_ASSISTANCE_TO_ATTACK)]
       [ACTION  '($SETQ NAVIGATOR_SEARCH_COMPLETED 'TRUE)]
    ]


    [RULE_N02

        [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
        [ACTION  '(
            ($PASCAL
              "fndng_clsest_base;
               fndng_frndly_AC;")
            ($SETQ CLOSEST_BASE_FOUND 'TRUE)
            ($SETQ FRIENDLY_AC_FOUND 'TRUE)
            ($SETQ NAVIGATOR_ASSISTANCE_TO_ATTACK 'COMPLETED))] ]


    [RULE_N03

        [PREMISE '($EQ INBOUND_ENEMY 'NONE)]
        [ACTION  '( ($SETQ NAVIGATOR_ASSISTANCE_TO_ATTACK 'NOT_NEEDED)
                    ($SETQ CLOSEST_BASE_FOUND 'FALSE)
                    ($SETQ FRIENDLY_AC_FOUND 'FALSE) )] ]]
```

Figure 7.5 Executive Knowledge Base

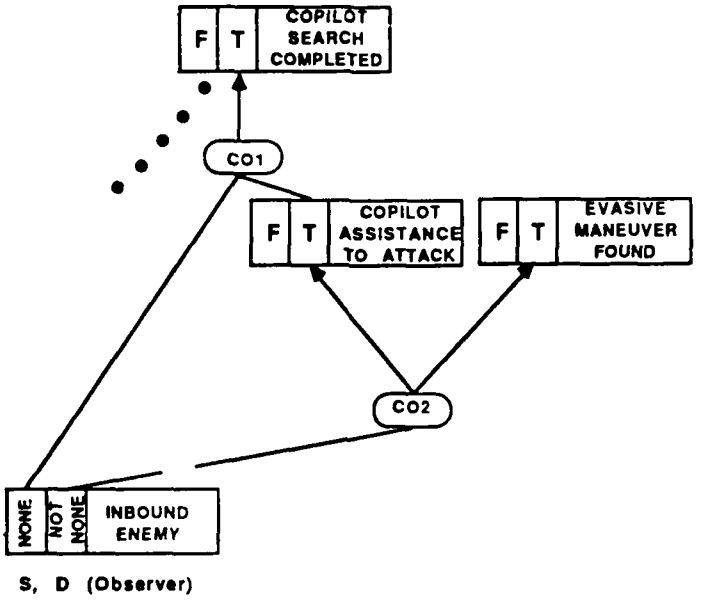


Listing 7.5 Executive Knowledge Base

```
~  executive kb for SAM attack demo
~  assumes executive gives clearance for defensive missile launch
~  assumes successful defense
~  shared parameters (with defender) are DEFEND_SUCCESS
~                                       LAUNCH_CLEARANCE
~                    (with observer)    INBOUND_ENEMY


~  (SETQ *SYSTEM_ID_CHARACTER* "X4")
~  (SETQ *EXPERT_ID* "executive")

~  executive  parameters


 [INTERN_SYMBOL_GROUP *EXECUTIVE_PARAMETERS*

    [EXECUTIVE_SEARCH_COMPLETED
                (EXPECT 'BOOLEAN)]

    [EXECUTIVE_ASSISTANCE_TO_ATTACK
                (EXPECT 'BOOLEAN)]

    [SITUATION_ANALYZED
                (EXPECT 'BOOLEAN)]

    [ADDITIONAL_TASKS_ORDERED
                (EXPECT 'BOOLEAN)]

    [SUCCESS_DETERMINED
                (EXPECT 'BOOLEAN)]

    [LAUNCH_CLEARANCE
                (EXPECT '(GO NOGO))]

 ]

~  executive rules

 [INTERN_SYMBOL_GROUP *EXECUTIVE_RULES*

    [RULE_X01

       [PREMISE '($OR  ($EQ INBOUND_ENEMY 'NONE)
                       ($AND ($EQ EXECUTIVE_ASSISTANCE_TO_ATTACK 'TRUE)
                             ($DETERMINE_VALUE_OF SUCCESS_DETERMINED)))]
       [ACTION  '($SETQ EXECUTIVE_SEARCH_COMPLETED 'TRUE)]
    ]


    [RULE_X02

       [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
       [ACTION  '(
        ($PASCAL
            "anlyzing_situtn;
             chckng_mssion_plan;")
        ($PASCAL
             "wts(''adtnal_tsks_ordrd'',1,new_line(executive),executive);")
```

```
        ($SETQ SITUATION_ANALYZED 'TRUE)
        ($SETQ ADDITIONAL_TASKS_ORDERED 'TRUE)
        ($SETQ EXECUTIVE_ASSISTANCE_TO_ATTACK 'TRUE) )] ]


      [RULE_X03
        [PREMISE '($UNKNOWN DEFEND_SUCCESS)]
        [ACTION  '($SETQ SUCCESS_DETERMINED 'FALSE)] ]


     [RULE_X04
       [PREMISE '($EQ DEFEND_SUCCESS 'TRUE)]
       [ACTION  '( ($PASCAL
         "gtng_dfns_results;
          wts(''successful defense'',1,new_line(executive),executive);
          wts(''good work!'',1,new_line(executive),executive);
          delay(1000);
          initialize_buffers(executive);
          off_screen(executive);")
          ($SETQ SUCCESS_DETERMINED 'TRUE) )]]


      [RULE_X05
        [PREMISE '($UNKNOWN LAUNCH_CLEARANCE)]
        [ACTION  '(
               ($PASCAL
                 "frmng_lnch_dcisin;
                  wts(''clrnc_for_lnch_GO!'',1,new_line(executive),
                                executive);")
               ($SETQ LAUNCH_CLEARANCE 'GO))] ]

   ]

~  (FIND_PARAMETER_RULE_RELATIONS *EXECUTIVE_PARAMETERS*  *EXECUTIVE_RULES*)
```

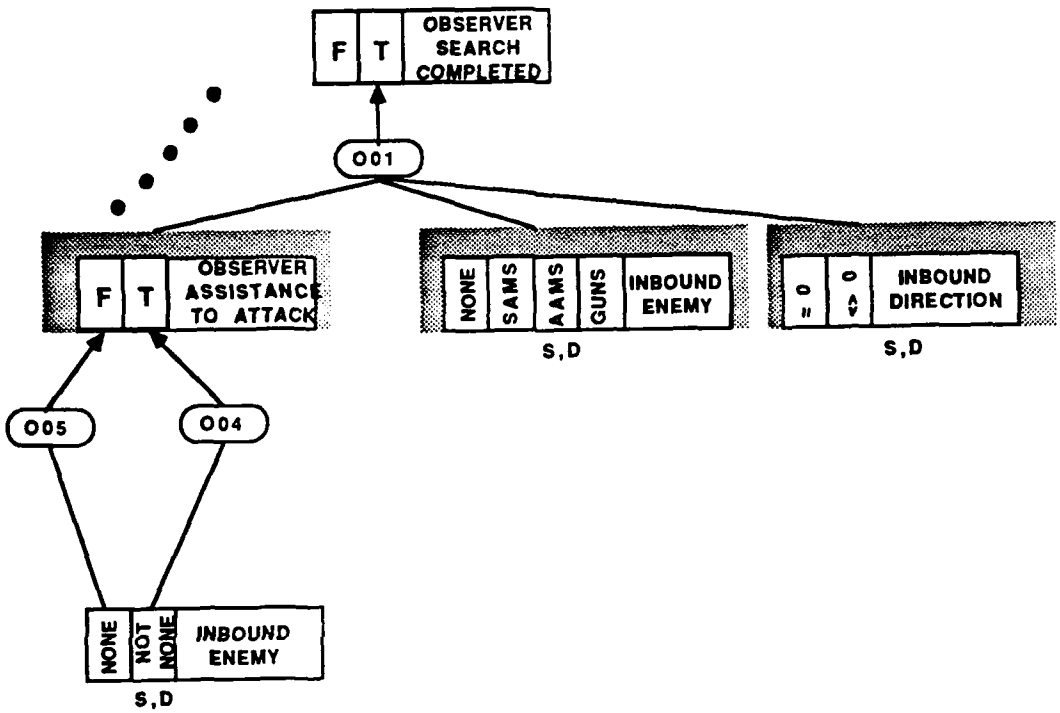
Figure 7.6 Copilot Knowledge Base



Listing 7.6 Copilot Knowledge Base

```
~  copilot kb for SAM attack demo
~
~  shared parameter (with defender/observer) is INBOUND_ENEMY
~

~
~  (SETQ *SYSTEM_ID_CHARACTER* "C4")
~  (SETQ *EXPERT_ID* "copilot")

~  copilot  parameters


 [INTERN_SYMBOL_GROUP *COPILOT_PARAMETERS*

     [COPILOT_SEARCH_COMPLETED
                  (EXPECT 'BOOLEAN)]

     [COPILOT_ASSISTANCE_TO_ATTACK
                  (EXPECT 'BOOLEAN)]

     [EVASIVE_MANEUVER_FOUND
                  (EXPECT 'BOOLEAN)]

 ]

~  copilot rules

  [INTERN_SYMBOL_GROUP *COPILOT_RULES*

     [RULE_C01

        [PREMISE '($OR  ($EQ INBOUND_ENEMY 'NONE)
                        ($EQ COPILOT_ASSISTANCE_TO_ATTACK 'TRUE) )]

        [ACTION  '($SETQ COPILOT_SEARCH_COMPLETED 'TRUE)]
     ]


     [RULE_C02

        [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
        [ACTION  '(
          ($PASCAL
             "fndng_evsve_mnvrs;")
          ($SETQ EVASIVE_MANEUVER_FOUND 'TRUE)
          ($SETQ COPILOT_ASSISTANCE_TO_ATTACK 'TRUE) )]
     ] ]


~  (FIND_PARAMETER_RULE_RELATIONS *COPILOT_PARAMETERS*  *COPILOT_RULES*)
```

Figure 7.7 Observer Knowledge Base

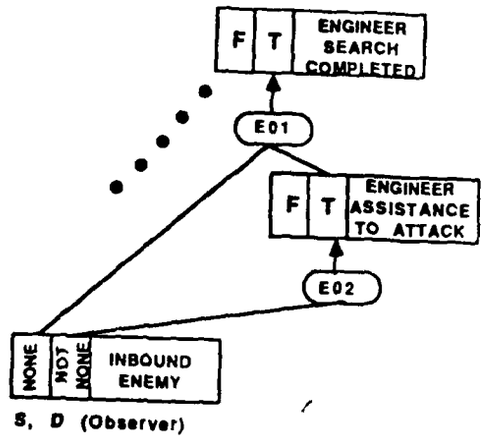


Listing 7.7 Observer Knowledge Base

```
~
~  observer kb for SAM attack demo
~
~  shared parameters (with eight remaining knowledge bases) are
~                        INBOUND_ENEMY
~                        INBOUND_DIRECTION
~
~

~
~ (SETQ *SYSTEM_ID_CHARACTER* "04")
~ (SETQ *EXPERT_ID* "observer")

~   observer  parameters


 [INTERN_SYMBOL_GROUP *OBSERVER_PARAMETERS*

     [OBSERVER_SEARCH_COMPLETED
                 (EXPECT 'BOOLEAN)]

     [OBSERVER_ASSISTANCE_TO_ATTACK
                 (EXPECT 'BOOLEAN)]

     [INBOUND_ENEMY
                 (EXPECT '(NONE, SAMS, AAMS, GUNFIRE)]

     [INBOUND_DIRECTION
                 (EXPECT 'INTEGER)]

  ]

~   observer rules

  [INTERN_SYMBOL_GROUP *OBSERVER_RULES*

     [RULE_001

        [PREMISE '($AND ($DETERMINE_VALUE_OF INBOUND_ENEMY)
                        ($DETERMINE_VALUE_OF OBSERVER_ASSISTANCE_TO_ATTACK))]
        [ACTION  '($SETQ OBSERVER_SEARCH_COMPLETED 'TRUE)]
     ]


     [RULE_002

        [PREMISE '($UNKNOWN INBOUND_ENEMY)]
        [ACTION  '(
          ($SETQ INBOUND_ENEMY 'NONE)
          ($SETQ INBOUND_DIRECTION '0)
          ($PASCAL
            "wts(''all clear'',1,new_line(observer),observer);
             get_key;"))] ]


      [RULE_003
          [PREMISE '($KNOWN INBOUND_ENEMY)]
          [ACTION '(
             ($PASCAL " get_key; ")
          ($SETQ INBOUND_ENEMY INBOUND_ENEMY)
```

```
        ($SETQ INBOUND_DIRECTION INBOUND_DIRECTION))] ]


      [RULE_004
         [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
         [ACTION  '(
           ($PASCAL "fndng_SAM_lnchpt;")
           ($SETQ OBSERVER_ASSISTANCE_TO_ATTACK 'TRUE))] ]


      [RULE_005
         [PREMISE '($EQ INBOUND_ENEMY 'NONE)]
         [ACTION  '($SETQ OBSERVER_ASSISTANCE_TO_ATTACK 'FALSE)] ]

    ]


~ (FIND_PARAMETER_RULE_RELATIONS *OBSERVER_PARAMETERS*  *OBSERVER_RULES*)
```

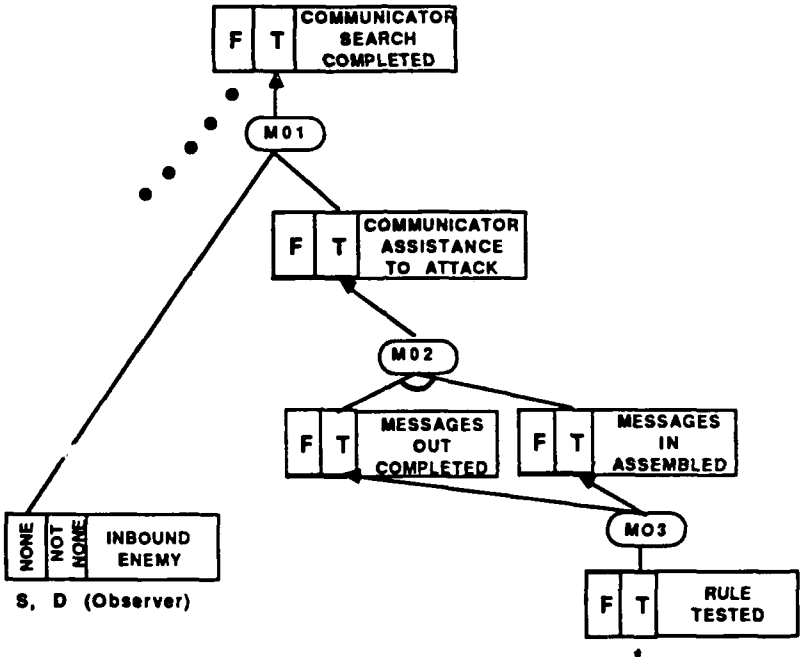
Figure 7.8 Engineer Knowledge Base



Listing 7.8 Engineer Knowledge Base

```
~  engineer kb for SAM attack demo
~
~  shared parameter (with defender/observer) is INBOUND_ENEMY

~ (SETQ *SYSTEM_ID_CHARACTER* "E4")
~ (SETQ *EXPERT_ID* "engineer")

~  engineer  parameters

 [INTERN_SYMBOL_GROUP *ENGINEER_PARAMETERS*

     [ENGINEER_SEARCH_COMPLETED
                  (EXPECT 'BOOLEAN)]

     [ENGINEER_ASSISTANCE_TO_ATTACK
                  (EXPECT 'BOOLEAN)]

 ]

~  engineer rules

  [INTERN_SYMBOL_GROUP *ENGINEER_RULES*

     [RULE_E01

        [PREMISE '($OR  ($EQ INBOUND_ENEMY 'NONE)
                        ($EQ ENGINEER_ASSISTANCE_TO_ATTACK 'TRUE) )]

        [ACTION  '($SETQ ENGINEER_SEARCH_COMPLETED 'TRUE)]
     ]

     [RULE_E02

        [PREMISE '($NOT ($EQ INBOUND_ENEMY 'NONE))]
        [ACTION  '(
          ($PASCAL
             "mntrng_systems;
              cmpting_AC_capablts;")
          ($SETQ ENGINEER_ASSISTANCE_TO_ATTACK 'TRUE) )]
     ] ]


~  (FIND_PARAMETER_RULE_RELATIONS *ENGINEER_PARAMETERS*  *ENGINEER_RULES*)
```

Figure 7.9 Communicator Knowledge Base



Listing 7.9 Communicator Knowledge Base

```
~
~  communicator kb for SAM attack demo
~
~  shared parameter (with defender/observer) is INBOUND_ENEMY

~
~ (SETQ *SYSTEM_ID_CHARACTER* "M4")
  (SETQ *EXPERT_ID* "communicator")

~   communicator  parameters


 [INTERN_SYMBOL_GROUP *COMMUNICATOR_PARAMETERS*

     [COMMUNICATOR_SEARCH_COMPLETED
                  (EXPECT 'BOOLEAN)]

     [COMMUNICATOR_ASSISTANCE_TO_ATTACK
                  (EXPECT 'BOOLEAN)]

     [MESSAGES_OUT_COMPLETED
                  (EXPECT 'BOOLEAN)]


     [MESSAGES_IN_ASSEMBLED
                  (EXPECT 'BOOLEAN)]

     [RULE_TESTED
                  (EXPECT 'BOOLEAN)
                  (INITIAL_VALUE 'TRUE)]

 ]


~   communicator rules

  [INTERN_SYMBOL_GROUP *COMMUNICATOR_RULES*

     [RULE_M01

        [PREMISE '($OR  ($EQ INBOUND_ENEMY 'NONE)
                        ($EQ COMMUNICATOR_ASSISTANCE_TO_ATTACK 'TRUE) )]

        [ACTION  '($SETQ COMMUNICATOR_SEARCH_COMPLETED 'TRUE)]
     ]

     [RULE_M02

        [PREMISE '($AND ($EQ MESSAGES_OUT_COMPLETED 'TRUE)
                        ($EQ MESSAGES_IN_ASSEMBLED 'TRUE)) ]
        [ACTION  '($SETQ COMMUNICATOR_ASSISTANCE_TO_ATTACK 'TRUE) ]
     ]

    [RULE_M03
      [PREMISE '($EQ RULE_TESTED 'TRUE)]
      [ACTION  '(
         ($PASCAL
         "messages_out;
          wts(''AC#3 under SAM atk'',1,new_line(communicator),communicator);
```

```
          messages_in;
          wts(''wngmt #1 cnfrms enmy'',1,new_line(communicator),communicator);")

          ($SETQ MESSAGES_OUT_COMPLETED 'TRUE)
          ($SETQ MESSAGES_IN_ASSEMBLED 'TRUE) )]

     ] ]

~  (FIND_PARAMETER_RULE_RELATIONS *COMMUNICATOR_PARAMETERS*  *COMMUNICATOR_RULES
```

8. Consolidation Of AUTOCREW Component Knowledge Bases Into One Process

AUTOCREW knowledge bases are developed and implemented individually. When a simulation process that includes all knowledge bases is developed, the rules and parameters of the individual knowledge bases must first be consolidated into single process lists. The resulting parameter-rule relationships represents a single process knowledge base containing all nine systems.

The following Princeton Rule-Based Controller (PRBC) [2] system software commands perform the tasks described. First, each AUTOCREW knowledge base is loaded into the LISP environment with the FLOAD command. The component knowledge-base parameters and rules are consolidated into larger lists with the CONSOLIDATE command. Finally, the process knowledge base is formulated with the FIND_PARAMETER_RULE_RELATIONS command.

```
~
~ this file consolidates all '4 lisp files to simulate 1 process
~
~ this simulation shows AUTOCREW response to an inbound SAM
~

   (SETQ *SYSTEM_ID_CHARACTER* "4")

~  attacker
   (FLOAD "KB_A4")

~  spoofer
   (FLOAD "KB_S4")

~  navigator
   (FLOAD "KB_N4")

~  copilot
   (FLOAD "KB_C4")

~  observer
   (FLOAD "KB_O4")

~  executive
   (FLOAD "KB_X4")

~  engineer
   (FLOAD "KB_E4")

~  communicator
   (FLOAD "KB_M4")

~  defender
   (FLOAD "KB_D4")

(FIND_PARAMETER_RULE_RELATIONS (CONSOLIDATE *ATTACKER_PARAMETERS*
                                            *SPOOFER_PARAMETERS*
                                            *NAVIGATOR_PARAMETERS*
                                            *COPILOT_PARAMETERS*
                                            *OBSERVER_PARAMETERS*
                                            *EXECUTIVE_PARAMETERS*
                                            *ENGINEER_PARAMETERS*
                                            *COMMUNICATOR_PARAMETERS*
                                            *DEFENDER_PARAMETERS*)

                               (CONSOLIDATE *ATTACKER_RULES*
                                            *SPOOFER_RULES*
                                            *NAVIGATOR_RULES*
                                            *COPILOT_RULES*
                                            *OBSERVER_RULES*
                                            *EXECUTIVE_RULES*
                                            *ENGINEER_RULES*
                                            *COMMUNICATOR_RULES*
                                            *DEFENDER_RULES*)

   )
```

9. Process Knowledge-Base Map

Graphical knowledge-base representation provides the best overview of knowledge-base logic and functions in comparison with code inspection. The PRBC development system facilitates logic tracing by providing a mapping procedure that shows how the parameters are set and used by rules. For example, the representation

```
5 ------------------------- ECM_DEPLOYMENT_COMPLETED ------------------- 5
    (15) RULE_S02 -> (* 4)
```

reads,

> "Rule S02 uses parameter #15 to set parameter ECM_DEPLOYMENT_COMPLETED and parameter #4"

The representation

```
4 ------------------------- SPOOFER_ASSISTANCE_TO_ATTACK ------------- 4
                              (*) RULE_SO1 -> (3)
```

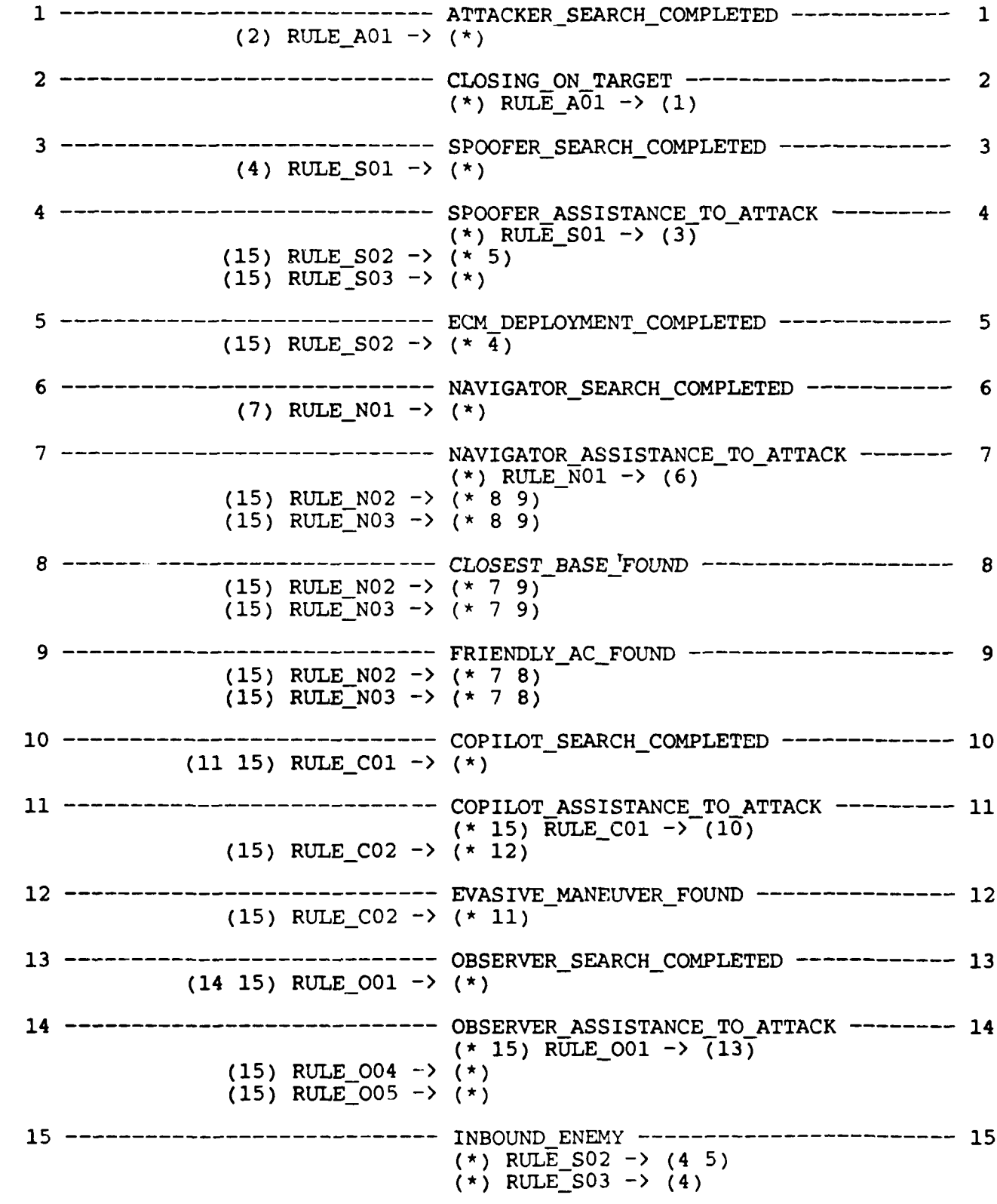
reads,

> the parameter SPOOFER_ASSISTANCE_TO_ATTACK is used by RULE S01 to set parameter #3"

The drawback of the PRBC representation is that the parameter values are invisible, so that a direct reconstruction of the rule is not possible. In contrast, the graphical representation facilitates direct reconstruction of the rules found in the knowledge-base code (or vice-versa). The advantage of the PRBC knowledge-base map representation is its provision of a condensed parameter/rule interaction "history".

The following PRBC output is a parameter/rule map of the inbound SAM attack process knowledge base.

Expert System 4 Knowledge Base Map



```
                                (*) RULE_N02 -> (7 8 9)
                                (*) RULE_N03 -> (7 8 9)
                                (* 11) RULE_C01 -> (10)
                                (*) RULE_C02 -> (11 12)
                                (* 14) RULE_O01 -> (13)
                                (*) RULE_O02 -> (* 16)
                                (*) RULE_O03 -> (* 16)
                                (*) RULE_O04 -> (14)
                                (*) RULE_O05 -> (14)
                                (* 18 21) RULE_X01 -> (17)
                                (*) RULE_X02 -> (18 19 20)
                                (* 24) RULE_E01 -> (23)
                                (*) RULE_E02 -> (24)
                                (* 26) RULE_M01 -> (25)
                                (* 31) RULE_D01 -> (30)
                                (* 33) RULE_D03 -> (31 32)
                                (*) RULE_O02 -> (* 16)
                                (*) RULE_O03 -> (* 16)
                    (33) RULE_D02 -> (* 31 32)

16 ---------------------------- INBOUND_DIRECTION ---------------------- 16
                                (*) RULE_D07 -> (34 39 40)
                    (15) RULE_O02 -> (* 15)
                    (15) RULE_O03 -> (* 15)

17 ---------------------------- EXECUTIVE_SEARCH_COMPLETED ------------- 17
             (15 18 21) RULE_X01 -> (*)

18 ---------------------------- EXECUTIVE_ASSISTANCE_TO_ATTACK --------- 18
                                (* 15 21) RULE_X01 -> (17)
                    (15) RULE_X02 -> (* 19 20)

19 ---------------------------- SITUATION_ANALYZED --------------------- 19
                    (15) RULE_X02 -> (* 18 20)

20 ---------------------------- ADDITIONAL_TASKS_ORDERED --------------- 20
                    (15) RULE_X02 -> (* 18 19)

21 ---------------------------- SUCCESS_DETERMINED --------------------- 21
                                (* 15 18) RULE_X01 -> (17)
                    (31) RULE_X03 -> (*)
                    (31) RULE_X04 -> (*)

22 ---------------------------- LAUNCH_CLEARANCE ----------------------- 22
                                (*) RULE_X05 -> (*)
                                (*) RULE_X05 -> (*)

23 ---------------------------- ENGINEER_SEARCH_COMPLETED -------------- 23
                (15 24) RULE_E01 -> (*)

24 ---------------------------- ENGINEER_ASSISTANCE_TO_ATTACK ---------- 24
                                (* 15) RULE_E01 -> (23)
                    (15) RULE_E02 -> (*)

25 ---------------------------- COMMUNICATOR_SEARCH_COMPLETED ---------- 25
                (15 26) RULE_M01 -> (*)

26 ---------------------------- COMMUNICATOR_ASSISTANCE_TO_ATTACK ------ 26
```

```
                                (* 15) RULE_M01 -> (25)
            (27 28) RULE_M02 -> (*)

27 ---------------------------- MESSAGES_OUT_COMPLETED ---------------- 27
                                (* 28) RULE_M02 -> (26)
               (29) RULE_M03 -> (* 28)

28 ---------------------------- MESSAGES_IN_ASSEMBLED ----------------- 28
                                (* 27) RULE_M02 -> (26)
               (29) RULE_M03 -> (* 27)

29 ---------------------------- RULE_TESTED --------------------------- 29
                                (*) RULE_M03 -> (27 28)
                                (*) RULE_D06 -> (38)
                                (*) RULE_D08 -> (36 41)
                                (*) RULE_D10 -> (42)
                                (*) RULE_D11 -> (37)

30 ---------------------------- DEFENDER_SEARCH_COMPLETED ------------- 30
            (15 31) RULE_D01 -> (*)

31 ---------------------------- DEFEND_SUCCESS ------------------------ 31
                                (*) RULE_X03 -> (21)
                                (*) RULE_X04 -> (21)
                                (* 15) RULE_D01 -> (30)
               (33) RULE_D02 -> (* 15 32)
            (15 33) RULE_D03 -> (* 32)

32 ---------------------------- DEFEND_COUNTDOWN_RELEASE -------------- 32
               (33) RULE_D02 -> (* 15 31)
            (15 33) RULE_D03 -> (* 31)

33 ---------------------------- ATTACK_OR_ABORT ----------------------- 33
                                (*) RULE_D02 -> (15 31 32)
                                (* 15) RULE_D03 -> (31 32)
      (34 35 37 38) RULE_D04 -> (*)
               (38) RULE_D05 -> (*)

34 ---------------------------- TRAJECTORY_COMPUTED ------------------- 34
                                (* 35 37 38) RULE_D04 -> (33)
               (16) RULE_D07 -> (* 39 40)

35 ---------------------------- DEFEND_WEAPON_SELECTED ---------------- 35
                                (* 34 37 38) RULE_D04 -> (33)
            (41 42) RULE_D09 -> (*)

36 ---------------------------- WEAPON_RECOMMENDED -------------------- 36
               (29) RULE_D08 -> (* 41)

37 ---------------------------- DEFEND_RELEASE_POINT_COMPUTED --------- 37
                                (* 34 35 38) RULE_D04 -> (33)
               (29) RULE_D11 -> (*)

38 ---------------------------- PILOT_CONFIRMED_ENEMY ----------------- 38
                                (* 34 35 37) RULE_D04 -> (33)
                                (*) RULE_D05 -> (33)
               (29) RULE_D06 -> (*)
```

```
39 ---------------------------- DEFEND_FIRE_CONTROL_STATUS ----------- 39
                (16) RULE_D07 -> (* 34 40)

40 ---------------------------- DEFEND_TRACK_SCAN -------------------- 40
                (16) RULE_D07 -> (* 34 39)

41 ---------------------------- PILOT_CONFIRMS_WEAPON ---------------- 41
                                 (* 42) RULE_D09 -> (35)
                (29) RULE_D08 -> (* 36)

42 ---------------------------- PILOT_SELECTS_WEAPON ----------------- 42
                                 (* 41) RULE_D09 -> (35)
                (29) RULE_D10 -> (*)
```

10. PASCAL Implementation of Inbound SAM Attack Simulation

When the process knowledge-base has been formulated in the LISP environment, it can be translated into the PASCAL environment using the PRBC development system software. A PASCAL implementation of the depth-first search inference engine then operates on the translated knowledge base. The rule-based logic flow proceeds in the same manner as in the LISP environment albeit faster and without LISP-related process interruptions such as garbage collection. Additionally, PASCAL code or procedure calls embedded in the knowledge base may be executed. Hence, numerical code is developed in the naturally procedural language PASCAL, and the knowledge base is developed in the naturally symbolic language LISP. The ease with which PASCAL code can be embedded in the knowledge base structure makes the PRBC a highly suitable tool for task and algorithm scheduling.

The following PASCAL program listing shows the high-level implementation of the inbound SAM attack AUTOCREW simulation. The included files "KB_4.TYP" and "KB_4.VAR" contain the parameter/rule declaration and variable statements. The file "INFER.BB" contains utility functions and the PASCAL implementation of the inference engine. The next four included files contain the AUTOCREW simulation testbed code to drive the graphics, user-interface and programmed keys for simulation control. The included file "KB_4.INC" contains PASCAL dummy procedure shells formed by the PRBC PASCAL Tag Generator utility [1]. These program shells send messages to the AUTOCREW graphics interface describing the task to be performed by their implied procedure name. The task may not yet be coded, but the designer can still define the knowledge-base structure. Hence, rapid prototyping of cooperating rule-based systems is facilitated, and the code can be developed in depth at a more convenient time, when the designer is satisfied with a particular ensemble prototype. The remaining included files "KB_4.PRE" and "KB_4.ACT" contain PASCAL translations of the premise (if) and action (then) statements respectively which are found in the process knowledge base.

Before search may commence, the parameters must be initialized (ikb command). Without initialization, the inference engine cannot establish the state of the knowledge base and cannot proceed with the search. The top-level parameters of the OBSERVER and DEFENDER are searched. Upon completing a pass through each knowledge base, all parameters are reinitialized and the search commences again (repeat ikb ... until). Goal-directed search on the top-level parameters of the remaining AUTOCREW components is invoked from within the DEFENDER knowledge base in

this implementation. In a multi-processor simulation, each AUTOCREW crew member would reside on a dedicated processor and would implement goal-directed search on its own knowledge base. Hence a parallel implementation would not require a search call from within the DEFENDER knowledge base.

```
program expert_4;

label  start_here;


{$I \PRBC\BRENDA\KB_4.TYP     }
{$I \PRBC\BRENDA\KB_4.VAR     }
{$I \PRBC\BRENDA\INFER.BB     }
{$I \PRBC\BRENDA\AUTOCREW.VAR  }
{$I \PRBC\BRENDA\GRAPHICS.BB}
{$I \PRBC\BRENDA\EXSPECIF.BB }
{$I \PRBC\BRENDA\KEYS_D.BB }
{$I \PRBC\BRENDA\KB_4.INC     }
{$I \PRBC\BRENDA\KB_4.PRE     }
{$I \PRBC\BRENDA\KB_4.ACT     }



{#####   MAIN PROGRAM   ################################################}

begin

   verbose := false;
   if verbose = false then start_up_display;
   if verbose = true then window(1,1,80,25);
   read_knowledge_base;

start_here:  repeat
               ikb;
               dvo(observer_search_completed);
               dvo(defender_search_completed);
             until keystroke = helpp;
             get_help;
             goto start_here;

end.   {expert_4}
```

END

DATE

FILMED

DTIC

9-88